Scalable Computing with MapReduce

Doug Cutting
doug@nutch.org

3 August 2005
OSCON, Portland, OR

Background: Nutch is scalable

- Nutch is web search software
  - aims to be very scalable
  - dominated by sort-merge computations
    - updating url frontier when crawling
    - inverting links, etc.
- Initial implementation is scalable...
  - parallel processes on multiple machines
  - some serial bottlenecks, but w/ plans to resolve
  - 100M web pages demonstrated

... but not to billions of pages

- scales better than other open source options
  - used by Creative Commons, OSU, etc.
- but large installations are operationally onerous
  - manually monitoring multiple machines is painful
  - data-interchange and space-allocation difficult
- with single operator
  - hard to use more than a handful of machines
  - effectively limited to ~100M pages

nutch

Need a Distributed File System

- single, shared namespace
- fault-tolerant
  - disk failures
  - node failures
- use disks already on computing nodes
- scales
  - easy to add & remove disks from live system
  - (with 1000 disks, expect disk failure daily)

nutch

Google publishes GFS paper

- meets all our needs (not surprisingly)
- single *namenode*
  - maps name → <blockId>*
  - maps blockId → <host:port>$^{replication\_level}$
- many *datanodes*, one per disk generally
  - map blockId → <byte>*
  - poll namenode for replication, deletion, etc. requests
- client code talks to both

Nutch Distributed File System (NDFS)

- if it's good enough for Google...
- open-source, Java implementation of GFS
- part of Nutch project
- first implemented in 2003 by Mike Cafarella
- currently maintained in branches/mapred

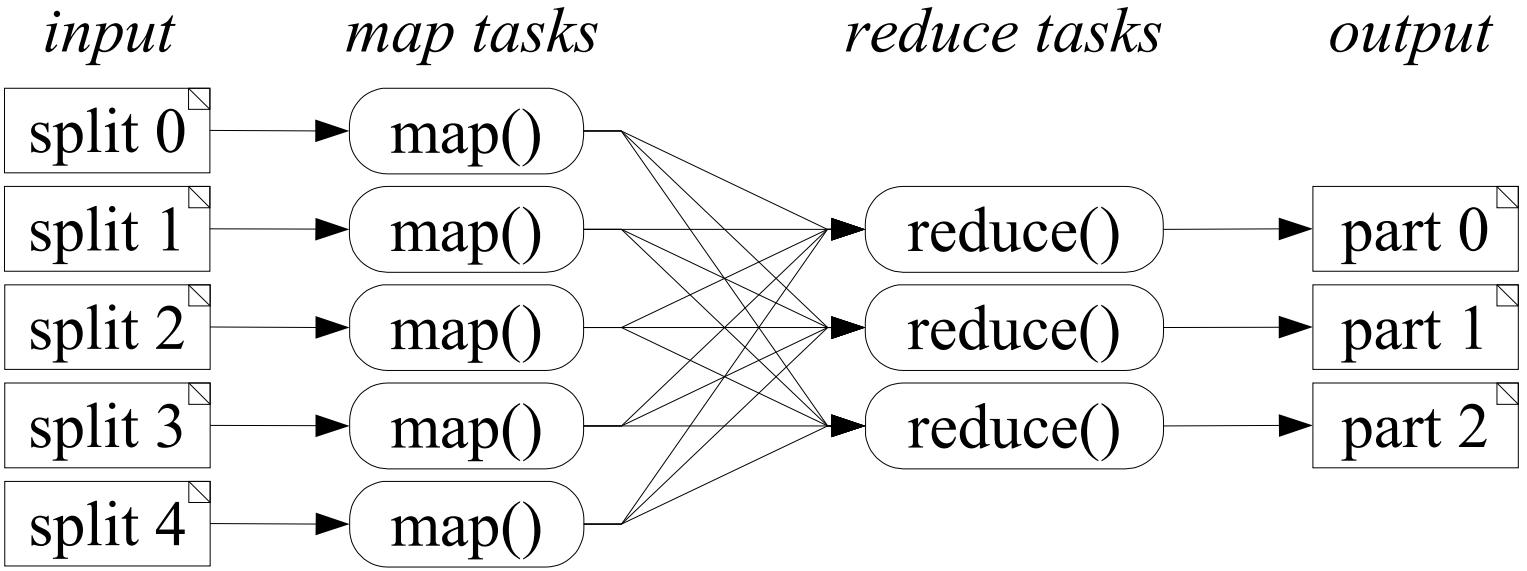
Need Distributed Computing Platform

- partition stages (*jobs)* into work units (*tasks)*
- for each task
  - allocate host
  - start
  - monitor
  - kill when hung
  - re-start when fails
- sequencing
  - start next job when one job is complete

Google publishes MapReduce paper

- Platform for reliable, scalable computing.
- All data is sequences of <key,value> pairs.
- Programmer specifies two primary methods:
  - map(k, v) → <k', v'>*
  - reduce(k', <v'>*) → <k', v'>*
  - also partition(), compare(), & others
- All v' with same k' are reduced together, in order.
  - bonus: built-in support for sort/merge!

MapReduce job processing



Use MapReduce in Nutch

- if it's good enough for Google...
  - trust that, while not ultimate arch., workable arch.
- not all Nutch computations are sort/merge
  - but all can easily be made to fit MapReduce
- not optimal for all computations
  - but not far off, and perhaps cheaper in the end
- reliable distributed platform tricky to develop
  - best to focus on a single implementation

Nutch on MapReduce

- Nutch's major algorithms converted in 2 weeks.
- Before:
  - several were undistributed scalabilty bottlenecks
  - distributable algorithms were complex to manage
  - collections larger than 100M pages impractical
- After:
  - all are scalable, distributed, easy to operate
  - code is substantially smaller & simpler
  - should permit multi-billion page collections

Nutch MapReduce Extensions

- Split output to multiple files
  - saves subsequent i/o, since inputs are smaller
- Mix input value types
  - saves MapReduce passes to convert values
- Async Map
  - permits multi-threaded Fetcher
- Partition by Value
  - facilitates selecting subsets w/ maximum key values

Example: RegexMapper

```
public class RegexMapper implements Mapper {
  private Pattern pattern;
  private int group;

  public void configure(JobConf job) {
    pattern = Pattern.compile(job.get("mapred.mapper.regex"));
    group = job.getInt("mapred.mapper.regex.group", 0);
  }

  public void map(WritableComparable key, Writable value,
                  OutputCollector output, Reporter reporter)
    throws IOException {
    String text = ((UTF8)value).toString();
    Matcher matcher = pattern.matcher(text);
    while (matcher.find()) {
      output.collect(new UTF8(matcher.group(group)),
                     new LongWritable(1));
    }
  }
}
```

Example: LongSumReducer

```
public class LongSumReducer implements Reducer {

  public void configure(JobConf job) {}

  public void reduce(WritableComparable key, Iterator values,
                     OutputCollector output, Reporter reporter)
    throws IOException {

    long sum = 0;

    while (values.hasNext()) {
      sum += ((LongWritable)values.next()).get();
    }

    output.collect(key, new LongWritable(sum));
  }
}
```

Example: main()

```
public static void main(String[] args) throws IOException {
  NutchConf defaults = NutchConf.get();
  JobConf job = new JobConf(defaults);

  job.setInputDir(new File(args[0]));

  job.setMapperClass(RegexMapper.class);
  job.set("mapred.mapper.regex", args[2]);
  job.set("mapred.mapper.regex.group", args[3]);

  job.setReducerClass(LongSumReducer.class);

  job.setOutputDir(args[1]);
  job.setOutputKeyClass(UTF8.class);
  job.setOutputValueClass(LongWritable.class);

  JobClient.runJob(job);
}
```

Nutch Configuration

- By default, configured for
  - in-process, single map & reduce task
  - local filesystem
- Config files loaded, in order:
  - nutch-default.xml – defaults (never edit)
  - mapred-default.xml – overrides to defaults
  - job.xml – specific to a MapReduce job
  - nutch-site.xml – cannot be overridden

NDFS configuration

- in nutch-site.xml:
  - specify namenode host, port & directory

```
<property>
  <name>fs.default.name</name>
  <value>server-01.domain.com:8009</value>
</property>
<property>
  <name>ndfs.name.dir</name>
  <value>/nutch/names</value>
</property>
```

  - specify location for files on each datanode

```
<property>
  <name>ndfs.data.dir</name>
  <value>/nutch/data</value>
</property>
```

MapReduce configuration

- in nutch-site.xml, specify job tracker host & port

```
<property>
  <name>mapred.job.tracker</name>
  <value>server-01.domain.com:8010</value>
</property>
```

- in mapred-default.xml, specify task numbers

```
<property>
  <name>mapred.map.tasks</name>
  <value>12</value>
</property>
<property>
  <name>mapred.reduce.tasks</name>
  <value>4</value>
</property>
```

Nutch @ Internet Archive

- given 40-node Capricorn rack July 1
  - 4 x 400GB drives per node
  - 1Ghz Via processor, 512MB RAM
- currently using for testing & debugging
- DNS & router issues have prevented large crawl
- hope to demonstrate 1B page index soon
  - crawling, and/or
  - indexing Archive's pre-crawled data

NDFS benchmark

- using 30-nodes, one drive only
- 10,000 files with random data
  - average 100MB, total 1TB
- 4 hours to create w/ replication=2
- 2 hours to read
- average 5MB/s per node, 1Gb/s overall
  - around 50% of optimal
  - scheduling is bottleneck, not disk, network or CPU

MapReduce status

- under active development
- usable today
- until Nutch 0.7 release, in branch:
  - https://svn.apache.org/repos/asf/lucene/nutch/branches/mapred/

Thanks!

Questions?

http://lucene.apache.org/nutch/